Table of Contents

Demonstration of controller synthesis via SReachPoint: Dubin's vehicle

This example will demonstrate the use of SReachTools in controller synthesis for stochastic continuous-state discrete-time linear time-varying (LTV) systems.

Specifically, we will discuss how we can use SReachPoint to synthesize open-loop controllers and affine-disturbance feedback controllers for that maximize the probability of safety while respecting the system dynamics and control bounds. We demonstrate: * `genzps-open`: Fourier transforms that uses Genz's algorithm to formulate a nonlinear log-concave optimization problem to be solved using MATLAB's patternsearch to synthesize an open-loop controller * `particle-open`: Particle control approach filter that formulates a mixed-integer linear program to synthesize an open-loop controller * `chance-open`: Chance-constrained approach that uses risk allocation and piecewise-affine approximations to formulate a linear program to synthesize an open-loop controller * `chance-affine`: Chance-constrained approach that uses risk allocation and piecewise-affine approximations to formulate a linear program to synthesize a closed-loop (affine disturbance feedback) controller

Our approaches are grid-free and recursion-free resulting in highly scalable solutions, especially for Gaussian-perturbed LTV systems.

```
% Prescript running
close all;clc;
clearvars;
srtinit
% Methods to run
ft_run = 0;
cc_open_run = 0;
cc_affine_run = 0;
pa_open_run = 0;
```

Problem: Stochastic reachability of a target tube for a Dubin's vehicle

We consider a Dubin's vehicle with known turning rate sequence $\overline{\omega} = \left[\omega_0\ \omega_1\ \ldots\ \omega_{T-1}\right]^\top \in R^T$, with additive Gaussian disturbance. The resulting dynamics are,

$$x_{k+1} = x_k + T_s \cos\left(\theta_0 + \sum_{i=1}^{k-1} \omega_i T_s\right) v_k$$

$$y_{k+1} = y_k + T_s \sin\left(\theta_0 + \sum_{i=1}^{k-1} \omega_i T_s\right) v_k$$

where x, y are the positions (state) of the Dubin's vehicle in x- and y- axes, v_k is the velocity of the vehicle (input), T_s is the sampling time, and θ_0 is the initial heading direction.

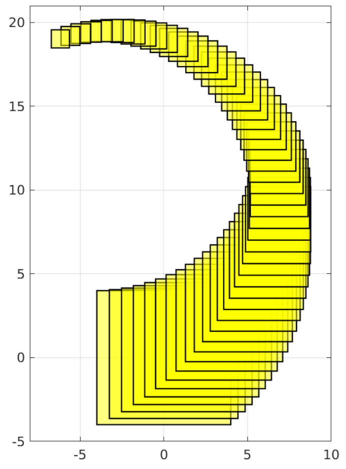
```
time_horizon = 50;
time_const = 1/2*time_horizon;
init_heading = pi/10;
sampling_time = 0.1;
box_halflength = 4;
omega = pi/time_horizon/sampling_time;
turning_rate = omega*ones(time_horizon,1);
dist_cov = 0.001;
v_nominal = 10;
umax = v_nominal/3*2;

[sys, heading_vec] = getDubinsCarLtv('add-dist', turning_rate,
 init_heading, ...
    sampling_time, Polyhedron('lb',0,'ub',umax), eye(2), ...
    RandomVector('Gaussian',zeros(2,1), dist_cov * eye(2)));
```

Target tube definition

```
target_tube_cell = cell(time_horizon + 1,1);
figure(100);clf;hold on
angle_at_the_center = (heading_vec) - pi/2;
center_box = zeros(2, time_horizon + 1);
for itt = 0:time_horizon
    center_box(:, itt+1) = v_nominal * [cos(angle_at_the_center(itt
+1))-cos(angle_at_the_center(1));
                                        sin(angle_at_the_center(itt
+1))-sin(angle_at_the_center(1))];
    target_tube_cell{itt+1} = Polyhedron('lb',center_box(:, itt+1) -
 box_halflength * exp(- itt/time_const), 'ub', center_box(:, itt+1) +
 box_halflength*exp(- itt/time_const));
    plot(target_tube_cell{itt+1},'alpha',0.5,'color','y');
end
axis equal
axis([-8    10   -5   21]);
```

```
box on;
grid on;
target_tube = Tube(target_tube_cell{:});
```



Initial states for each of the method

```
init_state_ccc_open = [2;2] + [-1;1];
init_state_genzps_open = [2;2] + [1;-1];
init_state_particle_open = [2;2] + [0;1];
init_state_ccc_affine = [2;2] + [2;1];
```

Quantities needed to compute the optimal mean trajectory

We first compute the dynamics of the concatenated state vector $X = Zx_0 + HU + GW$, and the compute the random vector GW and its mean.

```
[Z,H,G] = sys.getConcatMats(time_horizon);
GW = G * sys.dist.concat(time_horizon);
GW_mean = GW.parameters.mean;
```

SReachPoint: chance-open

This implements the chance-constrained approach to compute the optimal open-loop controller. This approach uses risk allocation and piecewise-affine overapproximation of the inverse normal cumulative density function to formulate a linear program for this purpose. Naturally, this is one of the fastest ways to compute an open-loop controller and an underapproximative probabilistic guarantee of safety.

```
if cc_open_run
    % Set the maximum piecewise-affine overapproximation error to 1e-3
    opts = SReachPointOptions('term', 'chance-
open','pwa_accuracy',1e-3);
```

```
    tic;
    [prob_ccc_open, opt_input_vec_ccc_open] = SReachPoint('term', ...
        'chance-open', sys, init_state_ccc_open, target_tube, opts);
    elapsed_time_ccc_open = toc;
    % Optimal mean trajectory construction
    optimal_mean_X_ccc_open = Z * init_state_ccc_open + ...
        H * opt_input_vec_ccc_open + GW_mean;
    optimal_mean_trajectory_ccc_open =
 reshape(optimal_mean_X_ccc_open, ...
        sys.state_dim,[]);
    disp(elapsed_time_ccc_open);
end
```

SReachPoint: chance-affine

This implements the chance-constrained approach to compute a locally optimal affine disturbance feedback controller. This approach uses risk allocation and piecewise-affine overapproximation of the inverse normal cumulative density function to formulate a difference-of-convex program. We utilize penalty convex-concave procedure to solve this program to a local optimum. Due to its construction of the affine feedback controller, this approach typically permits the construction of the highest underapproximative probability guarantee. Since affine disturbance feedback controllers can not satisfy hard control bounds, we relax the control bounds to be probabilistically violated with at most a probability of 0.01

```
if cc_affine_run
    opts = SReachPointOptions('term', 'chance-affine',...
        'max_input_viol_prob', 1e-2, 'verbose',2);
    tic
    [prob_ccc_affine, opt_input_vec_ccc_affine,
 opt_input_gain_ccc_affine] =...
        SReachPoint('term', 'chance-affine', sys,
 init_state_ccc_affine, ...
            target_tube, opts);
    elapsed_time_ccc_affine = toc;
    % Optimal mean trajectory construction
    % X = Z * x_0 + H * (M \mu_W + d) + G * \mu_W
    muW = sys.dist.concat(time_horizon).parameters.mean;
    optimal_mean_X_ccc_affine = Z * init_state_ccc_affine + H * ...
        (opt_input_gain_ccc_affine * muW + opt_input_vec_ccc_affine) +
 G * muW;
    optimal_mean_trajectory_ccc_affine =
 reshape(optimal_mean_X_ccc_affine, ...
        sys.state_dim,[]);
end
```

SReachPoint: genzps-open

```
if ft_run
    opts = SReachPointOptions('term', 'genzps-open', ...
        'PSoptions',psoptimset('display','iter'));
    tic
    [prob_genzps_open, opt_input_vec_genzps_open] =
 SReachPoint('term', ...
        'genzps-open', sys, init_state_genzps_open, target_tube,
 opts);
    elapsed_time_genzps = toc;
    % Optimal mean trajectory construction
    optimal_mean_X_genzps_open =  Z * init_state_genzps_open + ...
        H * opt_input_vec_genzps_open + GW_mean;
    optimal_mean_trajectory_genzps_open=
 reshape(optimal_mean_X_genzps_open, ...
        sys.state_dim,[]);
end
```

SReachPoint: particle-open (use verbosity 1)

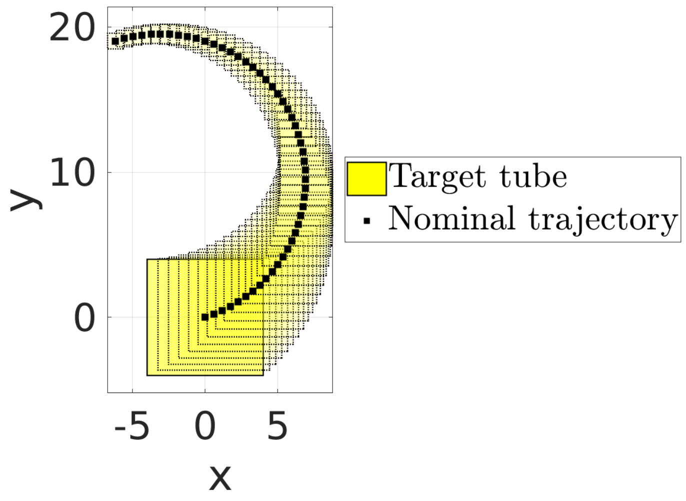
```
if pa_open_run
    opts = SReachPointOptions('term','particle-open','verbose',1,...
        'num_particles',50);
    tic
    [prob_particle_open, opt_input_vec_particle_open] =
 SReachPoint('term', ...
        'particle-open', sys, init_state_particle_open, target_tube,
 opts);
    elapsed_time_particle = toc;
    % Optimal mean trajectory construction
    optimal_mean_X_particle_open =  Z * init_state_particle_open + ...
        H * opt_input_vec_particle_open + GW_mean;
    optimal_mean_trajectory_particle_open =
 reshape(optimal_mean_X_particle_open, ...
        sys.state_dim,[]);
end
```

Plot the set

```
figure(101);
clf;
hold on;
for itt = 0:time_horizon
    if itt==0
        % Remember the first the tube
        h_target_tube =
 plot(target_tube_cell{1},'alpha',0.5,'color','y');
    else
        plot(target_tube_cell{itt
+1},'alpha',0.08,'LineStyle',':','color','y');
    end
end
axis equal
h_nominal_traj = scatter(center_box(1,:), center_box(2,:),
 50,'ks','filled');
h_vec = [h_target_tube, h_nominal_traj];
legend_cell = {'Target tube', 'Nominal trajectory'};
% Plot the optimal mean trajectory from the vertex under study
if cc_open_run
    h_opt_mean_ccc = scatter(...
            [init_state_ccc_open(1),
 optimal_mean_trajectory_ccc_open(1,:)], ...
            [init_state_ccc_open(2),
 optimal_mean_trajectory_ccc_open(2,:)], ...
            30, 'bo', 'filled','DisplayName', 'Mean trajectory (chance-
open)');
    legend_cell{end+1} = 'Mean trajectory (chance-open)';
    h_vec(end+1) = h_opt_mean_ccc;
end
if cc_affine_run
    h_opt_mean_ccc_affine = scatter(...
            [init_state_ccc_affine(1),
 optimal_mean_trajectory_ccc_affine(1,:)], ...
            [init_state_ccc_affine(2),
 optimal_mean_trajectory_ccc_affine(2,:)], ...
            30, 'ms', 'filled','DisplayName', 'Mean trajectory (chance-
affine)');
    legend_cell{end+1} = 'Mean trajectory (chance-affine)';
    h_vec(end+1) = h_opt_mean_ccc_affine;
end
if ft_run
    h_opt_mean_genzps = scatter(...
            [init_state_genzps_open(1),
 optimal_mean_trajectory_genzps_open(1,:)], ...
            [init_state_genzps_open(2),
 optimal_mean_trajectory_genzps_open(2,:)], ...
            30, 'kd','DisplayName', 'Mean trajectory (genzps-open)');
    legend_cell{end+1} = 'Mean trajectory (genzps-open)';
    h_vec(end+1) = h_opt_mean_genzps;
end
if pa_open_run
    h_opt_mean_particle = scatter(...
            [init_state_particle_open(1),
 optimal_mean_trajectory_particle_open(1,:)], ...
            [init_state_particle_open(2),
 optimal_mean_trajectory_particle_open(2,:)], ...
            30, 'r^', 'filled','DisplayName', 'Mean trajectory
 (particle-open)');
    legend_cell{end+1} = 'Mean trajectory (particle-open)';
    h_vec(end+1) = h_opt_mean_particle;
end
legend(h_vec,
 legend_cell, 'Location','EastOutside', 'interpreter','latex');
xlabel('x');
ylabel('y');
axis equal
box on;
set(gca,'FontSize',30);

% %% Check ccc-affine
% n_mcarlo_sims = 1e5;
% concat_state_realization = generateMonteCarloSims(n_mcarlo_sims, ...
%     sys, init_state_ccc_affine, time_horizon,
 opt_input_vec_ccc_affine, opt_input_gain_ccc_affine);
```

```
% mcarlo_result = target_tube.contains(concat_state_realization);
% fprintf('SReachPoint prob: %1.2f, Simulated prob: %1.2f',
 prob_ccc_affine, sum(mcarlo_result)/n_mcarlo_sims);
%
% %% Check particle-open
% n_mcarlo_sims = 1e5;
% concat_state_realization = generateMonteCarloSims(n_mcarlo_sims, ...
%     sys, init_state_particle_open, time_horizon,
 opt_input_vec_particle_open);
% mcarlo_result = target_tube.contains(concat_state_realization);
% fprintf('SReachPoint prob: %1.2f, Simulated prob: %1.2f',
 prob_particle_open, sum(mcarlo_result)/n_mcarlo_sims);
%
% Get a random point
% init_state_ccc_open = ccc_polytope.randomPoint();
```



Published with MATLAB® R2017b